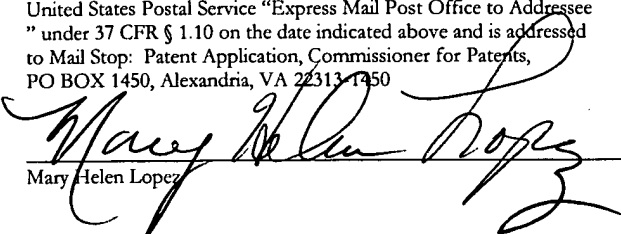


"Express Mail" mailing label number

EV 346024956 US

Date of Deposit: November 26, 2003

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" under 37 CFR § 1.10 on the date indicated above and is addressed to Mail Stop: Patent Application, Commissioner for Patents, PO BOX 1450, Alexandria, VA 22313-1450


Mary Helen Lopez

UNITED STATES PATENT APPLICATION

FOR

PARALLEL PIPELINE GRAPHICS SYSTEM

INVENTORS:

Mark M. Leather
Eric Demers

PREPARED BY:

Coudert Brothers LLP
333 South Hope Street
Twenty-Third Floor
Los Angeles, CA 90071
(213) 229-2900

This application claims priority to U.S. Provisional Application No. 60/429,976, filed November 27, 2002.

This is a related application to a co-pending U.S. Patent application entitled "DIVIDING WORK AMONG MULTIPLE GRAPHICS PIPELINES USING A SUPER-TILING

5 TECHNIQUE", having serial number 10/459,979, filed June 12, 2003, having Leather et al. as the inventors, owned by the same assignee and hereby incorporated by reference in its entirety.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates computer graphics chips.

10 Portions of the disclosure of this patent document contain material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever.

2. Background Art

15 Computer systems are often used to generate and display graphics on an output device such as a monitor. When complex and realistic graphics are desired there are often additional components, or chips, that are added to the computer system to assist it with the complex instruction processing that it must perform to render the graphics to the screen.

Graphics chips may be considered as having a front-end and a back-end. The front-end
20 typically receives graphics instructions and generates "primitives" that form the basis for the back-end's work. The back-end receives the primitives and performs the operations necessary to send the data to a frame buffer where it will eventually be rendered to the screen. As will be further

described below, graphics chip back-ends are currently inadequate. Before further discussing this problem, an overview of a graphics system is provided.

Graphics System

Display images are made up of thousands of tiny dots, where each dot is one of thousands
5 or millions of colors. These dots are known as picture elements, or "pixels". Each pixel has
multiple attributes associated with it, including a color and a texture which is represented by a
number value stored in the computer system. A three dimensional display image, although displayed
using a two dimensional array of pixels, may in fact be created by rendering of a plurality of
graphical objects. Examples of graphical objects include points, lines, polygons, and three
10 dimensional solid objects. Points, lines, and polygons represent rendering primitives which are the
basis for most rendering instructions. More complex structures, such as three dimensional objects,
are formed from a combination or mesh of such primitives. To display a particular scene, the visible
primitives associated with the scene are drawn individually by determining those pixels that fall
within the edges of the primitive, and obtaining the attributes of the primitive that correspond to
15 each of those pixels. The obtained attributes are used to determine the displayed color values of
applicable pixels.

Sometimes, a three dimensional display image is formed from overlapping primitives or
surfaces. A blending function based on an opacity value associated with each pixel of each primitive
is used to blend the colors of overlapping surfaces or layers when the top surface is not completely
20 opaque. The final displayed color of an individual pixel may thus be a blend of colors from multiple
surfaces or layers.

In some cases, graphical data is rendered by executing instructions from an application that
is drawing data to a display. During image rendering, three dimensional data is processed into a two

dimensional image suitable for display. The three dimensional image data represents attributes such as color, opacity, texture, depth, and perspective information. The draw commands from a program drawing to the display may include, for example, X and Y coordinates for the vertices of the primitive, as well as some attribute parameters for the primitive (color and depth or "Z" data), and a drawing command. The execution of drawing commands to generate a display image is known as graphics processing.

Graphics Processing Chips

When complex graphics processing is required, such as using primitives to as a basis for rendering instructions or texturing geometric patterns, graphics chips are added to the computer system. Graphics chips are specifically designed to handle the complex and tedious instruction processing that must be used to render the graphics to the screen. Graphics chips have a front-end and a back-end. The front-end typically receives graphics instructions and generates the primitives or combination of primitives that define geometric patterns.

The primitives are then processed by the back end where they might be textured, shaded, colored, or otherwise prepared for final output. When the primitives have been fully processed by the back end, the pixels on the screen will each have a specific number value that defines a unique color attribute the pixel will have when it is drawn. This final value is sent to a frame buffer in the back-end, where the value is used at the appropriate time.

Modern graphics processing chip back-ends are equipped to handle three-dimensional data, since three-dimensional data produces more realistic results to the screen. When processing three-dimensional data, memory bandwidth becomes a limitation on performance. The progression of graphics processing back-ends has been from a 32 bit system, to a 64 bit system, and to a 128 bit system. Moving to a 256 bit system, where 512 bits may be processed in a single logic clock cycle,

presents problems. In particular, the efficient organization and use of data “words” with a 256 bit wide DDR frame buffer is problematic because the granularity is too coarse. Increasing the width of the frame buffer to 256 bits requires innovations in the input and output (I/O) system used by the graphics processing back-end.

SUMMARY OF THE INVENTION

The present invention relates to a parallel array graphics system. In one embodiment, the parallel array graphics system includes a back-end configured to receive primitives and combinations of primitives (i.e., geometry) and process the geometry to produce values to place in a frame buffer for eventual rendering on a screen. In one embodiment, the graphics system includes two parallel pipelines. When data representing the geometry is presented to the back-end of the graphics chip, it is divided into data words and provided to one or both of the parallel pipelines.

In some embodiments, four parallel pipelines or other pipeline configurations having 2^n pipelines may be used. Each pipeline is a component of a raster back-end, where the display screen is divided into tiles and a defined portion of the screen (i.e., one or more tiles) is sent through a pipeline that owns that portion of the screen's tiles.

In one embodiment, each parallel pipeline comprises a raster back-end having a scan converter to step through the geometric patterns passed to the back-end, a "hierarchical-Z" component to more precisely define the borders of the geometry, a "Z-buffer" for performing three-dimensional operations on the data, a rasterizer for computing texture addresses and color components for a pixel, a unified shader for combining multiple characteristics for a pixel and outputting a single value, and a color buffer logic unit for taking the incoming shader color and blending it into the frame buffer using the current frame buffer blend operations. A plurality of FIFO (First-In, First-Out) units are used to balance load among the pipelines.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features, aspects and advantages of the present invention will become better understood with regard to the following description, appended claims and accompanying drawings

5 where:

Figure 1 is a parallel pipeline graphics system architecture according to an embodiment of the present invention.

Figure 2 is a flowchart showing the operation of a parallel pipeline graphics system according to an embodiment of the present invention.

10 Figure 3 is a parallel pipeline graphics system architecture according to another embodiment of the present invention.

Figure 4 is a flowchart showing the operation of a parallel pipeline graphics system according to another embodiment of the present invention.

15 Figure 5 is a raster back-end portion of a pipeline according to another embodiment of the present invention.

Figure 6 is a bounding box illustrating an embodiment of the invention.

Figure 7 shows an apparatus for synchronizing graphics data and state according to an embodiment of the present invention.

20 Figure 8 is an embodiment of a computer execution environment suitable for the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The invention relates to a parallel pipeline graphics system. In the following description, numerous specific details are set forth to provide a more thorough description of embodiments of the invention. It will be apparent, however, to one skilled in the art, that the invention may be practiced without these specific details. In other instances, well known features have not been described in detail so as not to obscure the invention.

Parallel Array Graphics System

One embodiment of the present invention is shown in the block diagram of Figure 1.

Graphics processing chip 100 comprises a front-end 110 and a back end 120. The front-end 110 receives graphics instructions 115 as input and generates geometry 116 as output. The back-end 120 is used to process the geometry 116 it receives as input. For instance, the back-end 120 might operate by texturing, shading, scanning, coloring, or otherwise preparing a pixel for final output.

When the geometry 116 has been fully processed by back-end 120, the pixels on the screen will each have a specific number value that defines a unique color attribute the pixel will have when it is drawn. The number values are passed to a frame buffer 130 where they are stored for use at the appropriate time, for instance, when they are rendered on display device 160. Back-end 120 includes two parallel pipelines, designated pipeline 140 and pipeline 150. When data representing the geometry is presented to the back-end 120 of the graphics chip, it is divided into data words and provided to one or both of the parallel pipelines 140 and 150.

Figure 2 provides a flowchart showing the operation of the architecture of Figure 1 according to an embodiment of the present invention. At step 200 a graphics chip front-end receives graphics instructions as input and generates geometry as output. At step 210, a graphics

chip back-end obtains the geometry as input. Next, it is determined which pipelines to use to operate on the geometry at step 220. At step 230 the appropriate pipelines operate on the geometry, for instance, the pipelines might texture, shade, scan, color, or otherwise preparing the geometry for final output. Then, at step 240, the numerical values that are associated with the pixels that define the geometry are put into a frame buffer. The size of the frame buffer may vary.

In another embodiment, 2 or more pipelines are used and each pipeline is a component of a raster back-end. The display screen is divided into tiles and a defined portion of the screen is sent (i.e., one or more tiles) through a pipeline that owns that portion of the screen's tiles. This embodiment is shown in Figure 3. Graphics processing chip 300 comprises a front-end 310 and a back-end 320. The front-end 310 receives graphics instructions 315 as input and generates geometry 316 as output. The back-end 320 is used to process the geometry 316 it receives as input. For instance, the back-end 320 might operate by texturing, shading, scanning, coloring, or otherwise preparing a pixel for final output.

When the geometry 316 has been fully processed by back-end 320, the pixels on the screen will each have a specific number value that defines a unique color attribute the pixel will have when it is drawn. The number values are passed to a frame buffer 330 where they are stored for use at the appropriate time, for instance, when they are rendered on display device 360. Back-end 320 includes 2^n parallel pipelines, designated pipeline 0 through pipeline $n-1$. When data representing the geometry is presented to the back-end 320 of the graphics chip 300, it is analyzed by back-end 320 to determine which geometry (or portions of geometry) fall within a given tile. For instance, if pipeline 0 owns tile 0 on display device 360, then the geometry in tile 0 is passed to pipeline 0.

Figure 4 provides a flowchart showing the operation of the architecture of Figure 3 according to an embodiment of the present invention. At step 400 a graphics chip front-end receives graphics instructions as input and generates geometry as output. At step 410, a graphics

chip back-end obtains the geometry as input. Next, at step 420 the back-end analyzes the geometry to determine which pipeline owns which portion of the geometry, for instance if a geometry falls within two tiles, then the geometry processing is divided among the pipelines that own those tiles. At step 430 the appropriate pipelines operate on the geometry, for instance, the pipelines might texture, shade, scan, color, or otherwise preparing the geometry for final output. Then, at step 440, the numerical values that are associated with the pixels that define the geometry are put into a frame buffer.

Embodiment of a Back-End Graphics Chip

In one embodiment, each parallel pipeline comprises a raster back-end having a scan converter to step through the geometric patterns passed to the back-end, a “hierarchical-Z” component to more precisely define the borders of the geometry, a “Z-buffer” for performing three-dimensional operations on the data, a rasterizer for computing texture addresses and color components for a pixel, a unified shader for combining multiple characteristics for a pixel and outputting a single value, and a color buffer logic unit for taking the incoming shader color and blending it into the frame buffer using the current frame buffer blend operations.

In operation, graphics assembly unit 510 takes transformed vertices data and assembles complete graphics primitives – triangles or parallelograms, for instance. A set-up unit 515 receives the data output from graphics assembly 510 and generates slope and initial value information for each of the texture address, color, or Z parameters associated with the primitive. The resulting set-up information is passed to 2 or more identical pipelines. In the current example there are two pipelines, pipeline 520 and pipeline 525, but the present invention contemplates any configuration of parallel pipelines. In this example, each pipeline 520 and 525 owns one-half of the screens pixels. Allocation of work between the pipelines is made based on a repeating square pixel, tile pattern. In

one embodiment, logic 530 in the set-up unit 515 intersects the graphics primitives with the repeating tile pattern such that a primitive is only sent to a pipeline if it is likely that it will result in the generation of covered pixels. The functionality of a setup unit is further described in commonly owned co-pending U.S. Patent Application entitled "Scalable Rasterizer Interpolator", with serial
5 number 10/xxx,xxx, filed December XX, 2003, and is hereby fully incorporated by reference.

In one embodiment of the present invention, the set-up unit manages the distribution of polygons to the pipelines. As noted above, the display is divided into multiple tiles and each pipeline is responsible for a subset of the tiles. It should be noted that any number of square or non-square tiles can be used in the present invention.

10 A polygon can be composed of 1, 2, or 3 vertices. Vertices are given by the graphics application currently executing on a host system. The vertices are converted from object space 3-dimensional homogeneous coordinate system to a display (screen) based coordinate system. This conversion can be done on the host processor or in a front end section of the graphics chip (i.e. vertex transformation). The screen based coordinate system has at least X and Y coordinates for
15 each vertex. The set-up unit 515 creates a bounding box based on the screen space X, Y coordinates of each vertex as shown in Figure 6. The bounding box is then compared against a current tile pattern. The tiling pattern is based on the number of graphics pipelines currently active. For example, in a two (A and B) pipeline system, the upper left and lower right pixel tiles of a four tile quad are assigned to pipeline A and the upper right and lower left tiles to pipeline B (or vice
20 versa). In a single pipeline system, all tiles are assigned to pipeline A. In one embodiment, the setup unit computes initial value (at vertex 0) and slopes for each of up to 42 parameters associated with the current graphics primitive.

The bounding boxes' four corners are mapped to the tile pattern, simply by discarding the lower bits of X & Y. The four corners map to the same or different tiles. If they all map to the same

tile, then only the pipeline that is associated with that tile receives the polygon. If it maps to only tiles that are associated with only one pipeline, then again only that pipeline receives the polygon. In one embodiment, if it maps to tiles that are associated with multiple pipelines, then the entire polygon is sent to all pipelines.

5 Each pipeline contains an input FIFO 535 used to balance the load over different pipelines. A scan converter 540 works in conjunction with Hierarchical Z interface of Z buffer logic 555 to step through the geometry (e.g., triangle or parallelogram) within the bounds of the pipeline's tile pattern. In one embodiment, initial stepping is performed at a coarse level. For each of the coarse level tiles, a minimum (i.e., closest) Z value is computed. This is compared with the farthest Z value
10 for the tile stored in a hierarchical-Z buffer 550. If the compare fails, the tile is rejected. The functionality of the scan converter and Hierarchical Z interface is further described in commonly owned co-pending U.S. Patent Application entitled "Scalable Rasterizer Interpolator", with serial number 10/xxx,xxx, filed December XX, 2003, and is hereby fully incorporated by reference.

15 The second section of the scan converter 540 works in conjunction with the Early Z interface of the Z buffer logic 550 to step through the coarse tile at a fine level. In one embodiment, the coarse tile is subdivided into 2x2 regions (called "quads"). For each quad, coverage and Z (depth) information is computed. A single bit mode register specifies where Z buffering takes place. If the current Z buffering mode is set to "early", each quad is passed to the Z
20 buffer 555 where its Z values are compared against the values stored in the Z buffer at that location. Z values for those covered pixels which "pass" the z compare, are written back into the Z buffer, and a modified coverage mask describing the result of the Z compare test is passed back to the scan converter 540. At this stage, those quads for which none of the covered pixels passed the Z compare test are discarded. The early Z functionality attempts to minimize the amount of

work applied by the unified shader and texture unit to quads which are not visible. The functionality of the scan converter and Early Z interface is further described in commonly owned co-pending U.S. Patent Application entitled "Scalable Rasterizer Interpolator", with serial number 10/xxx,xxx, filed December XX, 2003, and is hereby fully incorporated by reference.

5 Rasterizer 560 computes up to multiple sets of 2D or 3D perspective correct texture addresses and colors for each quad. The time taken to transfer data for each quad depends on the total number of texture addresses and colors required by that quad.

 A unified shader 570 works in conjunction with the texture unit 585 and applies a programmed sequence of instructions to the rasterized values. These instructions may involve
10 simple mathematical functions (add, multiply, etc.) and may also involve requests to the texture unit. A unified shader reads in rasterized texture addresses and colors, and applies a programmed sequence of instructions. A unified shader is so named because the functions of a traditional color
15 shader and a traditional texture address shader are combined into a single, unified shader. The unified shader performs both color shading and texture address shading. The conventional
distinction between shading operations (i.e., color texture map and coordinate texture map or color shading operation and texture address operation) is not handled by the use of separate shaders. In this way, any operation, be it for color shading or texture shading, may loop back into the shader and be combined with any other operation.

 The functionality of a unified shader is further described in commonly owned co-pending
20 U.S. Patent Application entitled "Unified Shader", with serial number 10/xxx,xxx, filed December XX, 2003, and is hereby fully incorporated by reference.

 The coverage and Z information generated by the scan converter 540, is passed to the Z buffer logic 555 via the Late Z interface. If the current Z buffering mode is set to "late", the Z values for the quad are compared against the values stored in the Z buffer at that location. Z values

for those covered pixels which "pass" the z compare are written back into the Z buffer. A modified coverage mask describing the result of the Z compare test is passed to the color buffer 590.

Although early Z operation is preferred for performance reasons, in certain situations the unified shader might modify the contents of the coverage mask (for example based on the value of the alpha channel), and in these cases the Z buffering mode will need to be set to "late".

Color buffer logic 590 takes the incoming shader color and blends it into the frame buffer using the current frame buffer blend operations. Blending only takes place for the covered sample (e.g., those with a corresponding 1 in the coverage mask).

10 Synchronizing Graphics Data and State

At each point in the graphics pipeline requiring state information, a state register and FIFO is added in one embodiment of the present invention. This embodiment is shown in Figure 7 where pipeline 1000 includes state registers 1010, 1015, and 1020 and FIFOs 1025, 1030, and 1035. The state registers 1010-1020 store all the state pertaining to the current stage in the graphics pipeline. It is loaded from the register interface.

Upon receipt of a "synchronize" bit from the register interface, the entire state for the graphics stage gets written into a FIFO. If the FIFO is empty, this is then immediately available for use by the graphics stage. If the FIFO is partially full, the state gets buffered for future use. A state change is enacted by sending a "synchronize" token down the graphics pipeline. This has the effect of "pop"ing a new set of state off the FIFO. If any of the state-change FIFO's are empty, that stage will stall the previous stage, and remain in a wait state until the FIFO contains valid state data. This prevents a graphics stage from using "garbage" or "expired" state. The register interface stalls if any

of the FIFO's are completely full during a cycle for which its "synchronize" bit is asserted. This prevents any of the FIFO's from overrunning and losing data.

Embodiment of Computer Execution Environment (Hardware)

5 An embodiment of the invention can be implemented as computer software in the form of computer readable program code executed in a general purpose computing environment such as environment 800 illustrated in Figure 8, or in the form of bytecode class files executable within a Java™ run time environment running in such an environment, or in the form of bytecodes running on a processor (or devices enabled to process bytecodes) existing in a distributed environment (e.g.,
10 one or more processors on a network). A keyboard 810 and mouse 811 are coupled to a system bus 818. The keyboard and mouse are for introducing user input to the computer system and communicating that user input to central processing unit (CPU) 813. Other suitable input devices may be used in addition to, or in place of, the mouse 811 and keyboard 810. I/O (input/output) unit 819 coupled to bi-directional system bus 818 represents such I/O elements as a printer, A/V
15 (audio/video) I/O, etc.

Computer 801 may include a communication interface 820 coupled to bus 818. Communication interface 820 provides a two-way data communication coupling via a network link 821 to a local network 822. For example, if communication interface 820 is an integrated services digital network (ISDN) card or a modem, communication interface 820 provides a data
20 communication connection to the corresponding type of telephone line, which comprises part of network link 821. If communication interface 820 is a local area network (LAN) card, communication interface 820 provides a data communication connection via network link 821 to a compatible LAN. Wireless links are also possible. In any such implementation, communication

interface 820 sends and receives electrical, electromagnetic or optical signals which carry digital data streams representing various types of information.

Network link 821 typically provides data communication through one or more networks to other data devices. For example, network link 821 may provide a connection through local network 822 to host 823 or to data equipment operated by ISP 824. ISP 824 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 825. Local network 822 and Internet 825 may use electrical, electromagnetic or optical signals which carry digital data streams. The signals through the various networks and the signals on network link 821 and through communication interface 820, which carry the digital data to and from computer 800, are exemplary forms of carrier waves transporting the information.

Processor 813 may reside wholly on client computer 801 or wholly on server 826 or processor 813 may have its computational power distributed between computer 801 and server 826. Server 826 symbolically is represented in Figure 8 as one unit, but server 826 can also be distributed between multiple "tiers". In one embodiment, server 826 comprises a middle and back tier where application logic executes in the middle tier and persistent data is obtained in the back tier. In the case where processor 813 resides wholly on server 826, the results of the computations performed by processor 813 are transmitted to computer 801 via Internet 825, Internet Service Provider (ISP) 824, local network 822 and communication interface 820. In this way, computer 801 is able to display the results of the computation to a user in the form of output.

Computer 801 includes a video memory 814, main memory 815 and mass storage 812, all coupled to bi-directional system bus 818 along with keyboard 810, mouse 811 and processor 813. As with processor 813, in various computing environments, main memory 815 and mass storage 812, can reside wholly on server 826 or computer 801, or they may be distributed between the two.

Examples of systems where processor 813, main memory 815, and mass storage 812 are distributed between computer 801 and server 826 include the thin-client computing architecture developed by Sun Microsystems, Inc., the palm pilot computing device and other personal digital assistants, Internet ready cellular phones and other Internet computing devices, and in platform independent
5 computing environments, such as those that utilize the Java technologies also developed by Sun Microsystems, Inc.

The mass storage 812 may include both fixed and removable media, such as magnetic, optical or magnetic optical storage systems or any other available mass storage technology. Bus 818 may contain, for example, thirty-two address lines for addressing video memory 814 or main
10 memory 815. The system bus 818 may also include, for example, a 32-bit data bus for transferring data between and among the components, such as processor 813, main memory 815, video memory 814 and mass storage 812. Alternatively, multiplex data/address lines maybe used instead of separate data and address lines.

In one embodiment of the invention, the processor 813 is a microprocessor manufactured
15 by Motorola, such as the 680X0 processor or a microprocessor manufactured by Intel, such as the 80X86, or Pentium processor, or a SPARC microprocessor from Sun Microsystems, Inc. However, any other suitable microprocessor or microcomputer may be utilized. Main memory 815 may be comprised of dynamic random access memory (DRAM). Video memory 814 may be a dual-ported video random access memory. One port of the video memory 814 may be coupled to video
20 amplifier 816. The video amplifier 816 maybe used to drive a display/output device 817, such as a cathode ray tube (CRT) raster monitor. Video amplifier 816 is well known in the art and may be implemented by any suitable apparatus. This circuitry converts pixel data stored in video memory 814 to a raster signal suitable for use by display/output device 817. Display/output device 817 maybe any type of monitor suitable for displaying graphic images.

Computer 801 can send messages and receive data, including program code, through the network(s), network link 821, and communication interface 820. In the Internet example, remote server computer 826 might transmit a requested code for an application program through Internet 825, ISP 824, local network 822 and communication interface 820. The received code may be
5 executed by processor 813 as it is received, and/or stored in mass storage 812, or other non-volatile storage for later execution. In this manner, computer 800 may obtain application code in the form of a carrier wave. Alternatively, remote server computer 826 may execute applications using processor 813, and utilize mass storage 812, and/or video memory 815. The results of the execution at server 826 are then transmitted through Internet 825, ISP 824, local network 822 and
10 communication interface 820. In this example, computer 801 performs only input and output functions.

Application code may be embodied in any form of computer program product. A computer program product comprises a medium configured to store or transport computer readable code, or in which computer readable code may be embedded. Some examples of computer program
15 products are CD-ROM disks, ROM cards, floppy disks, magnetic tapes, computer hard drives, servers on a network, and carrier waves.

The computer systems described above are for example only. An embodiment of the invention may be implemented in any type of computer system or programming or processing environment.

20 Thus, a parallel pipeline graphics system is described in conjunction with one or more specific embodiments. The invention is defined by the claims and their full scope of equivalents.